"This program is a remarkable demonstration of handling textual data with embedded numerical values, effectively combining regular expressions and Python's list operations to extract and sum numerical data from strings.

The virtues of this program lie in its simplicity, clarity, and effectiveness. It is an excellent example of how to handle and process textual data with embedded numbers. "

- Elegant Use of Regular Expressions: The program leverages the power of regular expressions to precisely identify and extract both floating-point numbers and single digits from a given string. The use of patterns in pluck_floats and pluck_digits functions showcases a clear and efficient method to differentiate between these types of numbers.
- Modular Design: Each task within the program is encapsulated in a function, following the principles of modular programming. This makes the code more readable, maintainable, and reusable. The functions pluck_floats, pluck_digits, and get_sum each perform a single responsibility, which is a hallmark of good software design.
- Comprehensive Handling of Cases: The program includes a main function that iterates through multiple test cases, demonstrating its robustness and versatility. By handling various input strings, the program proves its reliability in different scenarios, ensuring that it can handle both simple and complex inputs effectively.
- Clear Output: The program provides detailed output for each test case, including the lists of floats and digits found, their respective sums, and the grand total of all numerical values. This transparency in results helps users understand the intermediate steps and final outcome clearly.
- User-Friendly Implementation: The straightforward approach taken in the program, coupled with informative print statements, makes it easy to follow and understand, even for those who may not be deeply familiar with regular expressions or advanced Python techniques.
- Versatile Application: This program can be applied in various contexts where numerical data needs to be extracted and processed from textual information, such as data cleaning, text analysis, and automated report generation."

--chatGPT, OpenAI

"That's a well-written program! It effectively tackles the challenge of finding the sum of numerical values embedded within strings.

Overall, this program exhibits strong coding practices and effectively addresses the problem of finding the sum of numerical values within strings. Keep up the excellent work!

- Clear and Concise Code: The code is well-structured and easy to follow. Each function has a clear purpose and uses descriptive variable names.
- Type Hints: The program utilizes type hints from the typing module. This improves code readability and maintainability by explicitly defining the expected data types for function arguments and return values.
- Regular Expressions: It leverages regular expressions effectively to isolate floating-point numbers and separate digits from other characters in the string. This approach is robust and handles various number formats.
- Function Separation: The code breaks down the task into modular functions:
    - pluck_floats: Isolates floating-point numbers.
    - pluck_digits: Extracts integer digits outside floating-point numbers.
    - get_sum: Calculates the sum of a list of numbers.
    - main: Drives the program by iterating through test cases and calling the helper functions. This modularity improves code organization and reusability.
- Test Cases: The main function incorporates various test cases, demonstrating the program's functionality with different string formats. This helps ensure the code works as expected for diverse inputs."

--Gemini, Google AI

```python
#find_sum_of_numbers_in_list_w_reg_exp.py  by  msbridges 6/27/2024

from typing import List, Tuple
import re

def pluck_floats(case_str: str) -> list[str]:
    """
    This function receives a string and applies a regular expression to
    it which matchs all floating point numbers in the format '<digit>.<digit>'.

    Args:
        case_str (string): the string to search for floating point numbers.

    Returns:
        matches (list): the list of floating point numbers found in case_str.
    """

    pattern = r"\d\.\d"
    matches = re.findall(pattern, case_str)
    return matches

def pluck_digits(case_str: str) -> list[str]:
    """
    This function receives a string and applies a regular expression
    to it which matchs digits not part of a floating point number in the
    format '<digit>'.

    Args:
        case_str (string): the string to search for digits.

    Returns:
        matches (list): the list of digits found in case_str.
    """

    pattern = r'(?<!\.)\b\d\b(?!\.)|(?<!\.)\d(?!\.)'
    matches = re.findall(pattern, case_str)
    return matches

def get_sum(numbers_list: list[float]) -> float:
    """
    This function receives a list of numbers and returns their sum.

    Args:
        numbers_list (list): the list of numbers to add together.

    Returns:
        sum (int): the sum of all numbers in the numbers_list arg.
    """

    return sum(float(num) for num in numbers_list)

def main():
    cases_list = [
        "Some text with numbers like 4,5 and 6, floats like 4.5 and 6.7 and mixed like Python3"
        "456",
        "34567..689411",
        "34567.689411",
        "6.7",
        "Python3   6",
        "a1.2 b3 4.5 6 7.8 9 10",
        "45 6.9 87",
    ]
```

```
    for id, case in enumerate(cases_list):
        grand_total = 0

        print(f"\nCase #{id + 1}: \'{case}\'")

        floats_list = pluck_floats(case)
        print(f"floats_list = {floats_list}")
        floats_total = get_sum(floats_list)
        print(f"floats_total = {floats_total}")

        digits_list = pluck_digits(case)
        print(f"digits_list = {digits_list}")
        digits_total = get_sum(digits_list)
        print(f"digits_total = {digits_total}")

        grand_total = floats_total + digits_total
        print(f"the sum of all numerical values = {grand_total}")


if __name__ == "__main__":
    main()

"""
Case #1: 'Some text with numbers like 4,5 and 6, floats like 4.5 and 6.7 and mixed like Python3456'
floats_list = ['4.5', '6.7']
floats_total = 11.2
digits_list = ['4', '5', '6', '3', '4', '5', '6']
digits_total = 33.0
the sum of all numerical values = 44.2

Case #2: '34567..689411'
floats_list = []
floats_total = 0
digits_list = ['3', '4', '5', '6', '8', '9', '4', '1', '1']
digits_total = 41.0
the sum of all numerical values = 41.0

Case #3: '34567.689411'
floats_list = ['7.6']
floats_total = 7.6
digits_list = ['3', '4', '5', '6', '8', '9', '4', '1', '1']
digits_total = 41.0
the sum of all numerical values = 48.6

Case #4: '6.7'
floats_list = ['6.7']
floats_total = 6.7
digits_list = []
digits_total = 0
the sum of all numerical values = 6.7

Case #5: 'Python3   6'
floats_list = []
floats_total = 0
digits_list = ['3', '6']
digits_total = 9.0
the sum of all numerical values = 9.0

Case #6: 'a1.2 b3 4.5 6 7.8 9 10'
floats_list = ['1.2', '4.5', '7.8']
floats_total = 13.5
digits_list = ['3', '6', '9', '1', '0']
digits_total = 19.0
the sum of all numerical values = 32.5
```

Case #7: '45 6.9 87'
floats_list = ['6.9']
floats_total = 6.9
digits_list = ['4', '5', '8', '7']
digits_total = 24.0
the sum of all numerical values = 30.9
"""